Por la presente declaro que esta propuesta es producto de mi propio trabajo y hasta donde sé y creo, no contiene material que de forma sustancial haya sido previamente publicado o escrito por otra persona ni material que haya sido aceptado para la concesión de premios de cualquier otro grado o diploma de la universidad u otro instituto de docencia superior, excepto donde se ha hecho el reconocimineto debido al texto

*(**signatura/ nom/data**).*

Escola Politècnica Superior
d'Enginyeria de Vilanova i la Geltrú
UNIVERSITAT POLITÈCNICA DE CATALUNYA

PROJECTE FI DE CARRERA

TÍTOL: MIBODA.COM

AUTOR: **Mari Luz Rodríguez Salas**

TITULACIÓ: **Ingeniería Técnica en Informática de Gestión**

DIRECTOR: José Luís Balcázar Navarro

DEPARTAMENT: LSI Llenguatges i Sistemes Informàtics

DATA: 29 Junio 2007

TÍTOL: MIBODA.COM

COGNOMS:	**Rodríguez Salas**	**NOM:**	**Mª Luz**
TITULACIÓ:	**Enginyeria Tca. Informàtica**		
ESPECIALITAT:	**Gestió**	**PLA: 93**	

DIRECTOR:	**José Luis Balcázar**
DEPARTAMENT:	**Llenguatges i Sistemes Informàtics (LSI)**

QUALIFICACIÓ DEL PFC

TRIBUNAL

PRESIDENT	SECRETARI	VOCAL
JOSE ANTONIO ROMAN JIMENEZ	JOAN VICENC GOMEZ URGELLES	JAVIER NAVARRO BOSQUE

DATA DE LECTURA: **12 Julio 2007**

Este Proyecto tiene en cuenta los aspectos medioambientales: No [X] □Sí

PROYECTO FIN DE CARRERA

RESUMEN

Miboda.com es un proyecto que surgió de la necesidad de un caso real: la organización de una boda.
El requerimiento principal era poder contar con una plataforma para interactuar, de forma fácil, con todos los invitados, ofreciéndoles una adaptabilidad y una "liberación" a los novios. Un medio para facilitar un feedback de información a cualquier hora. Información permanentemente expuesta, respuesta permanentemente emitible. Lo que nos proporciona un modelo de comunicación perfecto, en el que el elemento *canal* está siempre a nuestra disposición para que, tanto el *emisor* como el *receptor,* reciban el *mensaje* sin necesidad de que estén presentes en el momento de emitirlo: un *medio* estable, perenne y con capacidad para almacenar el mensaje.

El medio que nos va a ofrecer estas facilidades consiste en una aplicación Web con el objetivo de organizar y gestionar todo aquello referente a una boda.

El principal objetivo ha sido contruir una aplicación autogestionable por el usuario y totalmente personalizada sin la necesidad de intervención de ningún administrador, consiguiendo así un resultado totalmente personalizado.

Con esta pretensión era evidente que era necesario una Web dinámica que fuese capaz de distinguir diferentes perfiles de usuario (novios e inivitados) y de almacenar datos, es decir que accediera a una base de datos.

El resultado ha sido un portal desde el que los novios crean y gestionan su boda ya que ofrece una interacción novios - invitados bidireccional. Los novios pueden crear la lista de invitados, la lista de bodas, exponer toda aquella información que consideren de interés para los invitados y hacer un seguimiento estadístico. Los invitados por su parte podrán confirmar su asistencia, consultar, reservar o comprar regalos expuestos en la lista de bodas, dejar mensajes a los novios o consultar la información necesaria.

Para conseguir montar una aplicación estable y fácilmente escalble, que ofrezca la posibilidad de ofrecer estas funcionalidades, he trabajado sobre una plataforma compuesta por Tomcat(sevidor), struts(framework de j2ee) y mysql como modelo de base de datos.

Paraules clau :

Boda	Portal	Struts	Tomcat
Nuvis	Web	mariluz	miboda.com

Reconocimientos y agradecimientos

Quisiera darle primeramente las gracias a mi ponente José Luis Balcázar, por sacar tiempo de donde no lo había, por el apoyo y por la confianza depositada en mí.

También dar las gracias a Carlos y Marian, que fueron los culpables de impulsarme a embarcar en la locura de struts (de lo que hoy día tanto me alegro), a Rubén y Jonatan por su incondicional apoyo, a Manolo que me tendió la mano cuando más lo necesitaba y a Helena por su tiempo e interés constante.

... y, como no, a mis padres, cuyo afán por facilitarme todo, fue de vital importancia al principio y muy especialmente, a Juan Carlos, mi marido, el cliente al que me refiero durante todo el proyecto, no por nada, sino por todo.

A todos ellos, gracias por su granito de arena sin el que este proyecto no habría sido posible.

Índice de Contenidos

Parte I

Introducción y Metodología

Capítulo

1 Introducción

Nuestra sociedad se ha ido adaptando a las nuevas tecnologías en todos los ámbitos. Las gestiones bancarias se pueden realizar desde casa, sin adaptarse a los reducidos horarios de las entidades bancarias. Cualquier tipo de documentación e información que se pueda necesitar también lo tenemos al alcance en la red sin necesidad de acudir a una biblioteca, incluso las tareas más cotidianas como el realizar la compra diaria se pueden realizar desde casa; podemos llenar nuestro carrito de la compra sin necesidad de soportar largas colas en los supermercados, y todo eso desde casa y a cualquier hora.

Si los medios y la tecnología existe y está a nuestro alcance, porqué no aplicarlos para adaptar también tradiciones a los tiempos de hoy. Una boda comporta planificar, organizar y coordinar un sinfín de gestiones y preparativos en los que participa mucha gente. Y cada vez más todos los preparativos, típicos de una boda, se están haciendo extensivos a otro tipo de celebraciones como comuniones o bautizos. Es una realidad la tendencia a montar grandes eventos de lo que tradicionalmente tan sólo eran actos religiosos.

Esta tendencia a dar cada vez más importancia a la celebración y menos a la ceremonia religiosa en sí, ha ido evolucionando hacia lo material no sólo en el caso de las bodas sino también en comuniones y bautizos. En este último caso, en muchas ocasiones, ni si quiera llega a celebrarse el sacramento del bautizo sino que, directamente, se pasa a la parte consumista y material. Se organizan grandes banquetes a cambio de los cuales se esperan grandes regalos.

Debido a esta moda, con más auge cada vez, las tiendas ofrecen listas de regalos para recién nacidos y niños de comunión. Sin embargo, parece ser que no se ha pensado en atender la otra parte de la necesidad creada: organizar y gestionar los grandes eventos que se montan y facilitar gestionar estas listas de regalos desde casa.

1.1 Motivación

Un día, alguien con planes de boda, me planteó su necesidad y me preguntó si era un proyecto factible. Estuvimos comentando cuáles eran los aspectos que quería que cubriera el proyecto y hacia donde quería dirigirlo; qué funcionalidades, en líneas generales, consideraba que debía tener el proyecto y por lo tanto cuáles eran las necesidades que consideraba que necesitaba cubrir.

Estuve navegando algunos días buscando páginas relacionadas y realmente no encontré nada que cubriera las necesidades que esta persona me planteaba. Así consideré que era una buena idea (todavía no sabía como la llevaría a cabo, pero ya lo estudiaría), y que

además no quedaría en un proyecto teórico o académico, sino que sería un proyecto llevado a la práctica que sería de utilidad para alguien.

1.2 Situación actual

Actualmente existen un buen número de Webs que tratan la temática de las bodas. Esta variedad de páginas está experimentando un rápido crecimiento en los dos últimos años (desde que me planteé el tema de mi proyecto); sin embargo, si no todos, prácticamente la totalidad, son de contenido informativo y de referencia. He estado visitando varias Web que responden a criterios de búsqueda de *boda*, *organización boda*, *ceremonia*, *enlace matrimonial*, *novios*, etc. y todas las que he encontrado han sido informativas, sin ninguna operatividad propia.

La oferta actual de páginas que traten esta temática consiste en páginas comerciales: aquellas que ofrecen sus servicios para organizar tu boda (buscan restaurante, te organizan la despedida de novios, el viaje, etc.), otras se limitan a ofrecer un directorio de links a otras Web más especializadas que ofrecen sus productos (Web de vestidos de novias, de floristerías, de imprentas, de detalles de boda, etc.) y por último también existen el Web tipo revista electrónica, con artículos y reportajes con consejos e información sobre las tradiciones y el protocolo a seguir.

Algunas de estas páginas son www.bodas.org, www.guianupcial.es, www.publiboda.com, www.mifuturaboda.com y muchas más, sin embargo ninguna te ayuda a gestionar y te ofrece un servicio en sí misma sin remitirte a otras con fin comercial.

1.3 Objetivos

El objetivo ha sido construir una herramienta que facilite gestionar la organización de una boda. Una aplicación que sea un servicio tanto para los organizadores (los novios) como para los invitados, de forma personalizada.

Con esa pretensión surge la idea de este proyecto: construir un portal Web personalizado a cada pareja y dedicado a la gestión de su boda. Eliminar la necesidad de los novios de confeccionar reiteradas listas de invitados (frecuentemente inexactas), realizar y recibir incesantes llamadas (en ocasiones inoportunas), para solicitar confirmación de asistencia, suprimir el incómodo momento de preguntar y contestar qué regalar, en el caso de necesidad de hotel para invitados y/o de autocar, confirmar también el número de plazas necesarias sin tachones en la lista, ni equivocaciones, ni recuentos continuos, mantener informados puntualmente a los invitados de la fecha, los horarios, localizaciones y cuanta información se considere necesaria.

En general se pretende conseguir un servicio no sólo para facilitar a los novios la organización de la ceremonia, sino también a los invitados su participación en todo aquello que sea requerido.

Capítulo

2 Metodología

Ante la perspectiva de tener que afrontar un proyecto desde cero, debemos ser metodológicos y avanzar ordenadamente para no caer en una espiral de tener que retroceder ya en momento de pruebas a la parte inicial de establecer los requisitos.
Con esta pretensión utilizaremos un método ingenieril que nos ayudará a conseguir los objetivos deseados para nuestro programa:

- económico: buena relación calidad precio
- fiable: que los resultados sean los buscados
- eficiente: una buena relación entre recursos y velocidad resultante
- necesidades de los usuarios: que el resultado satisfaga las expectativas del cliente

El punto de partida para desglosar la globalidad del proyecto que nos hemos propuesto será separar la definición del problema del desarrollo:

2.1 Punto de partida

El punto de partida para desglosar la globalidad del proyecto (sistema) que nos hemos propuesto, será separar la definición del problema del desarrollo:

Definición del problema:

- Analizar el sistema (la globalidad del entorno)
- Planificación
- Determinar los requisitos

Desarrollo:

- Especificación
- Diseño
- Implementación: codificar
- Testeo: fase de pruebas

y en el caso de llegar a sacar este software a la calle habría que añadir una tercera fase:

Mantenimiento:

- Corregir
- Adaptar
- Mejorar

Éste fue el planteamiento inicial para ir avanzando en la creación del proyecto pero a medida que fui avanzando en la fase de programación te das cuenta que, tal y como ya está documentado en muchos libros, el ciclo de vida clásico es un modelo teórico difícil de llevar a la práctica porque:

- no es factible llevar a cabo un modelo estrictamente secuencial
- es imposible determinar al inicio exactamente los requisitos y que se puedan llevar a cabo sin tenerlos que modificar
- porque siguiendo el modelo teórico el producto sólo se obtendría al final del ciclo.

2.2 Modelo de ciclo de vida del proyecto

El ciclo de vida software es "el periodo que comienza cuando un producto software es concebido y termina cuando deja de estar disponible".

Diferentes modelos determinan de forma diferente el conjunto de actividades que se han de realizar durante el proceso así como el orden en que se realizan dichas actividades.

Por lo que decíamos en el apartado anterior el ciclo de vida de este proyecto se ha acercado más al de prototipaje (ver *Fig. 1*), cuya característica principal es contar con un prototipo, de rápida confección sobre el que se trabaja para tener una aproximación del producto final que se quiere conseguir.

En este proyecto se creó un prototipo navegable con powerpoint, muy rudimentario pero que proporcionaba lo que se pretendía de un prototipo:

- validación de la interfaz
- validación de los requisitos: contrastar con el cliente que esos eran los objetivos que él solicitaba
- validación de partes del sistema final

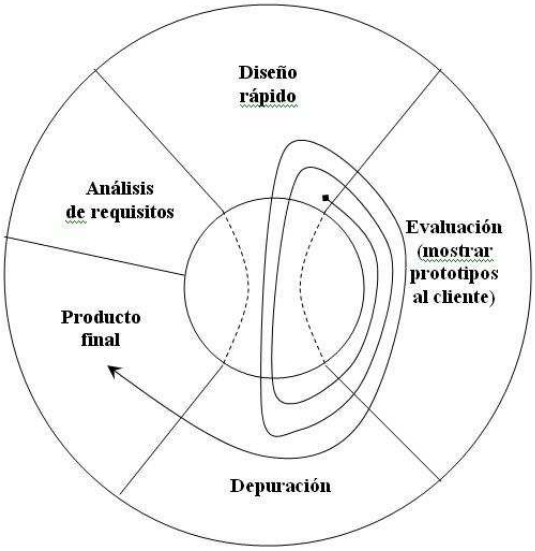

Fig. 1 *Gráfico del ciclo de vida de prototipaje*

Parte II

Proyecto

Capítulo

3 Modelo de desarrollo del Software

El proceso de desarrollo de software se puede determinar definiendo un pequeño número de actividades que son aplicables a todos los proyectos de software, con independencia de su tamaño y complejidad.
Para definir el proceso de desarrollo software estas actividades se pueden agrupar en tres fases genéricas:

- **Fase de definición:** se centra en el qué. Incluye tareas como el análisis del sistema como un conjunto, la planificación del proyecto y el análisis de requisitos.

- **Fase de desarrollo:** se centra en el cómo. Incluye tareas como el diseño, la generación de código y las pruebas de ejecución.

- **Fase de mantenimiento:** se centra en el cambio. Incluye tareas como la corrección, mejoras y adaptación.

Cuando nos encontramos ante un nuevo proyecto que emprender, se debe incorporar una estrategia que acompañe al proceso y seleccionar qué tareas y en qué orden, llevaremos a cabo. Esta estrategia a adoptar se denomina *modelo de desarrollo del proceso*.
Para seleccionar un modelo u otro hay que tener en cuenta la naturaleza del proyecto, los métodos y herramientas a utilizar y el seguimiento o control que se vaya a hacer del proyecto (entregas parciales, posibilidad de reuniones de seguimiento, etc).

El *ciclo de vida software* determina el orden en que se realizan las actividades. Es el periodo que comienza cuando un producto software es concebido y termina cuando deja de estar disponible.

Los principales *modelos de ciclo de vida software* son:

- ciclo de vida secuencial
- ciclo de vida clásico
- ciclo de vida en espiral
- ciclo de vida de prototipaje (de desarrollo de prototipos)

Capítulo

4 Especificación y Desarrollo

En la fase de especificación se hace un análisis detallado de “el qué”. El primer paso será hacer un análisis de requisitos, formalizaremos también la estructura de las tablas y entidades que tiene nuestro sistema con un modelo de datos: el de entidad-relación posteriormente haremos un modelo de clases.

Es de vital importancia (aunque no siempre posible) determinar exactamente cuáles son las expectativas del cliente antes de empezar, si quiera, con la fase de diseño. Pasos en falso nos harían retroceder y cambiar cosas que nos afectarían incluso a nivel de programación los cual nos supondría un bajo rendimiento y por tanto alto coste.
Para hacer un buen análisis de requisitos deberemos tener en cuenta los siguientes factores y puntos a seguir:

- Comprender los objetivos y necesidades del cliente: tener la mayor claridad posible de a dónde queremos llegar
- Determinar los conjuntos de sistemas que resuelven el problema: cuántos y qué factores serían necesarios para ofrecer una solución
- Evaluar las opciones posibles: en un principio no podremos descartar ninguna, pero sí evaluarlas para tener una idea de los pros y contras que ofrecen cada alternativa
- Escoger una opción

4.1 Una primera aproximación a los requisitos[1]

Tras haber aceptado el proyecto propuesto por el cliente el siguiente paso fue establecer en líneas generales cuales eran sus objetivos. Requería un sistema que le ayudara a controlar y gestionar de forma automática, tantos aspectos como fuera posible relacionados con la organización de su boda. Algunas de las funcionalidades básicas que debía cumplir eran:

- los datos no debían estar accesibles a todo el mundo sino sólo a las personas invitadas y a los propios novios.

- había que hacer tratamiento diferenciado en caso de que el usuario fuera *pareja* o *invitado*

- la información compartida por *novios* e *invitados* debía mostrarse a los invitados de forma filtrada (no se podía mostrar en la totalidad con la que se mostraba a

[1] Ver glosario de términos en apéndice

los novios)

- debe tener la capacidad de crear una lista de bodas y almacenarla
- esta lista de bodas tenía que poder modificarse durante todo el periodo de utilización por los novios
- debía ofrecer también la posibilidad de crear y modificar permanentemente una lista de invitados
- los invitados debían poder confirmar su asistencia
- junto con la asistencia, éstos debían confirmar también la plaza para el autocar
- había que habilitar algún apartado en el que los novios dejasen sus dedicatorias
- y muy importante: la finalidad debía ser informativa, tantas áreas o apartados como las necesidades fueran demandando

4.2 Análisis de requisitos

4.2.1 Requisitos funcionales

Son aquellos que detallan qué hace el sistema, cuáles serán las entradas y salidas (ej. mostrar la lista de artículos o introducir el nombre usuario para logarnos) y cuál será la relación entre éstas.

Código	Nombre requerimiento y descripción	Observaciones
RFAUTEN	*Autenticación* Que el sistema requiera un usuario registrado para poder entrar en él, y que esta autenticación hiciera un tratamiento diferenciado a los diferentes tipos de usuario (invitados y novios)	
RFBIENV	*Bienvenida* El portal debe ofrece un saludo personalizado al usuario autenticado	

RFESTBO	*Estado boda*[2] Mantener durante la conexión al usuario informado de la boda sobre la que está trabajando	
RFESTUS	*Estado usuario* Mantener durante la conexión los datos del usuario que se ha logado	
RFMENOP	*Menú opciones* Ofrecer un menú de operaciones diferenciado para cada tipo de usuario, ocultando a los invitados información privada reservada a los novios.	
ESPECÍFICOS PARA PERFIL DE USUARIOS "PAREJA"		
RFCRELB	*Crear lista de bodas* Crear la lista de bodas introduciendo uno a uno los artículos	
RFEDALB	*Editar articulo* Poder modificar cualquiera de los artículos introducidos en la lista de bodas en todo momento	
RFELALB	*Eliminar artículo* Suprimir el artículo deseado de la lista de bodas	
RFCREIN	*Crear lista invitados* Poder crear una lista a través de ir añadiendo cualquiera de los artículos introducidos en la lista de bodas en todo momento	Para poder añadir invitado, el sistema necesitará que primero se cree un usuario ya que el usuario es el que se loga.
RFELALB	*Eliminar invitado* Suprimir el invitado deseado de la lista de bodas	Cuando el inivtado a suprimir sea el último de ese usuario, el sistema eliminará también el usuario.
RFCONIN	*Consultar lista de invitados* Mostrar lista de invitados con toda la información relativa a la organización	Debe mostrar la información que es de interés para el cliente
RFCONCA	*Consultar canciones*	La función sera diferente ya

[2] Las aplicaciones web se basan en peticiones y respuestas por lo que sería muy costoso y difícil mantener un estado. En este tipo de aplicaciones cuando se habla de mantener el estado en verdad consiste en guardar en la sesión los atributos requeridos durante la conexión.

	Listar la información referente a las canciones recomendadas	que obtendrá más información que la llamada por un invitado.
RFADDIN	*Datos celebración* Insertar en la base de datos la información a mostrar después de donde será	
RFCREUS	*Crear usuario* La pareja deberá dar de alta a los usuarios que desee que entren en el sistema	
ESPECÍFICOS PARA PERFIL DE USUARIOS "INVITADO"		
RFGESRE	*Gestionar regalos* Seleccionar y "comprar" artículos de la lista de bodas	La información mostrada para los invitados no mostrará el nombre del comprador en aquellos artículos que aparezcan como no disponibles
RFDEJDE	*Dejar Dedicatoria* Dejar dedicatoria a pareja	
RFCAS	*Confirmar Asistencia* Confirmar la asistencia o la no asistencia al evento	
RFCAU	*Confirma autocar* Confirmar si requerirá reserva de plaza para el autocar.	
RFCCA	*Consultar Canciones* Listar las canciones actualmente ya recomendadas	
RFCAN	*Dejar Canción* Recomendar una canción para la celebración	
RFCONIN	*Consultar Información* Consultar información expuesta	
RFRESRE	*Reservar Regalo* Reservar o marcar como comprado el regalo seleccionado de la lista de bodas	

4.2.2 Requisitos no funcionales

Hace referencia a la “forma”, es decir, no se centra en el qué debe hacer el sistema sino en el cómo, en cuanto a aspectos externos (sin referirnos a la programación). Así en requisitos no funcionales se deben recoger aspectos como fiabilidad, velocidad, etc.

El cliente pidió que se pusiera especial atención en conseguir un sistema:

- intuitivo: que fuera de fácil navegación y uso sin necesidad de poner ayudas y que fuera incluso utilizable por personas poco habituadas a utilizar Internet

- rápido: que (teniendo en cuenta los medios con los que trabajo) consiguiera una navegación ágil, sin páginas que tardaran mucho en cargar

- eficaz: que las distintas opciones disponibles se pudieran llevar a cabo sin necesidad de dar muchos paso y/o navegar a través de muchas pantallas

4.3 Modelo de Datos

Un modelo de datos es un sistema formal y abstracto que permite describir los datos de acuerdo con reglas y convenios predefinidos. Es formal pues los objetos del sistema se manipulan siguiendo reglas perfectamente definidas y utilizando exclusivamente los operadores definidos en el sistema, independientemente de lo que estos objetos y operadores puedan significar.[3]

He elegido el modelo entidad-relación[4] (E/R) para representarlo. Éste es uno de los varios modelos conceptuales existentes para el diseño de bases de datos. Fue inventado por Peter Chen en los años setenta. El propósito de este modelo es simplificar el diseño de bases de datos a partir de descripciones textuales de los requisitos.

Los elementos esenciales del modelo son:

- **entidades**: Una entidad es un objeto que existe y que es distinguible de otros objetos. Existen entidades concretas y abstractas
- **atributos**: las entidades tienen atributos, éstos son características de la entidad, algunos son compartidos por otras entidades y otros son exclusivos
- **relaciones**: es una asociación entre entidades (normalmente dos)

4.3.1 Diagrama Entidad-Relación

[3] Ullman1999
Ullman, Jeffrey y **Widom, Jennifer**, *Introducción a los Sistemas de Bases de Datos*. Editorial Prentice Hall, México 1999, ISBN: 970-17-0256-5.

[4] Ver glosario de términos en el apéndice

Ver fig.2

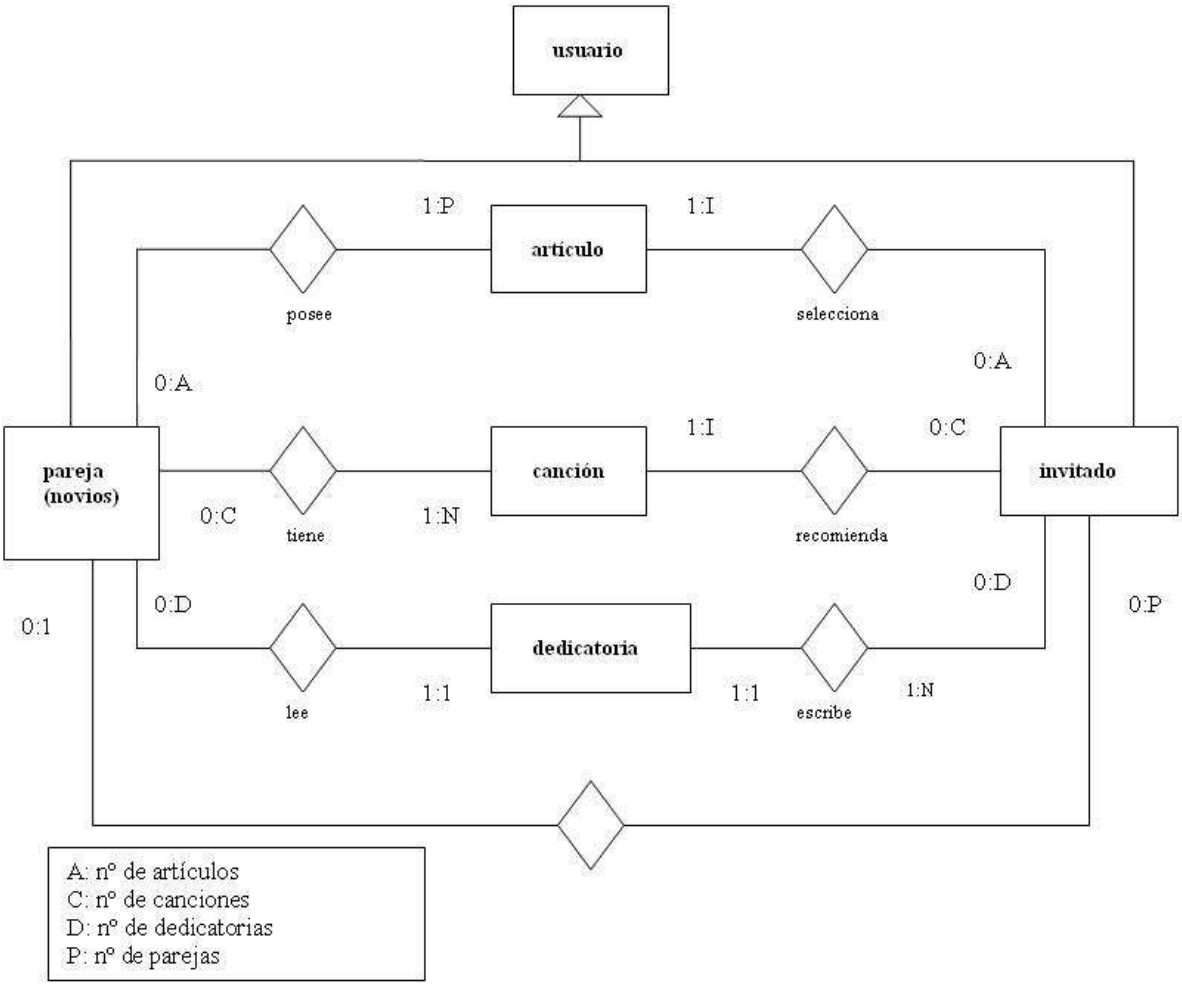

Fig.2 *Diagrama Entidad-Relacion E/R*

4.4 Diagrama de clases

En la figura 2 se muestra el diagrama de clases completo en el que se representan las clases que intervienen en nuestro proyecto a nivel de modelo[5] y la relación entre ellos.

[5] Este concepto se entenderá mejor una vez leído el apartado en el que se explica detalladamente struts.

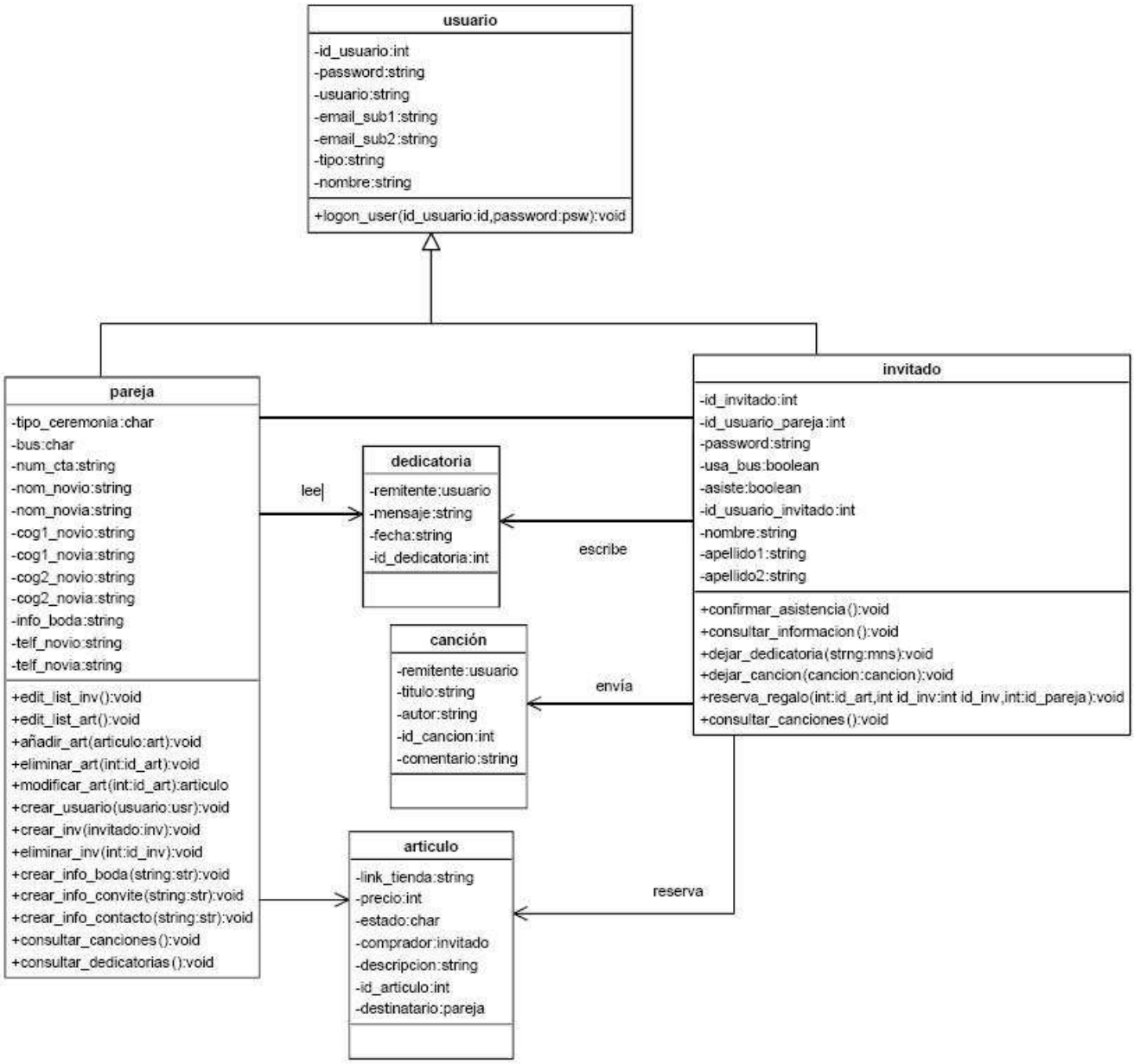

Fig2 *Diagrama de clases*

4.5 Diagrama de casos de uso

En esta aplicación sólo existen dos tipos de usuarios: el invitado y la pareja (o novios), por lo que sólo serán necesarios dos diagramas (ver *Fig.3* y *Fig.4*)para representar los casos de uso de los diferentes actores.

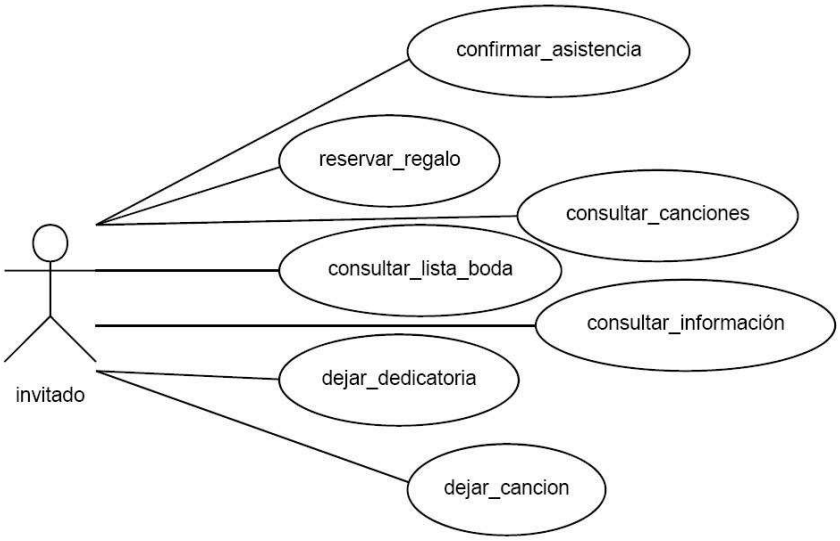

Fig 3 *Diagrama de caso de uso del invitado*

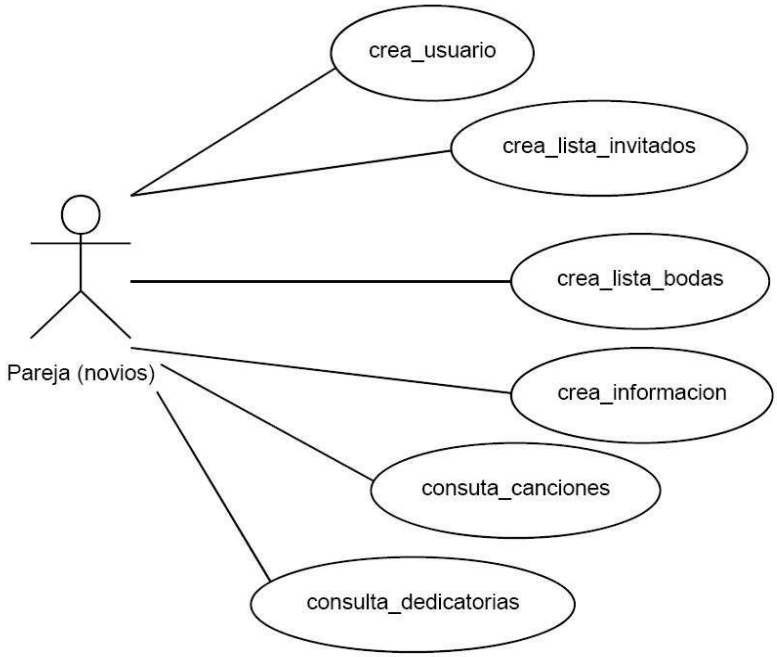

Fig 4 *Diagrama de caso de uso de pareja*

Capítulo

5 Implementación

La fase de implementación es el momento en el que todos los datos teóricos y abstractos hasta ahora detallados toman forma en algo concreto.

Una vez definidos los requisitos el siguiente paso era decidir como iba a representar y dar forma a todas estas ideas y objetivos.

5.1 Plataforma

Inicialmente era un proyecto que se pensaba implementar sobre LAMP (linux apache mysql y php) pero por diversos motivos, detallados en los siguiente apartados, se decidió pasar a Windows, Tomcat, Mysql y J2EE.
Windows era mucho más práctico a la hora de poder avanzar en mi proyecto en cualquier ordenador, Tomcat era el servidor que me permitía trabajar con struts, mysql era un un SGBD que me ofrecía una buena relación prestaciones/consumo de recursos teniendo en cuenta el hardware con el que contaba y J2EE como lenguaje de programación orientado a objetos y transportable a diferentes sistemas.

5.1.1 Base de datos (mysql)

Su principal objetivo de diseño fue la velocidad, por ello se suprimieron algunas características de los demás SGBDs (Sistema de Gestion de Base de Datos).

Ventajas:

- Mayor rendimiento
- Mejores utilidades de administración
- Integración perfecta con PHP
- Sin límites en los tamaños de los registros
- Mejor control de acceso de usuarios

En el caso de **MySQL**, el servidor es el que realiza todas las operaciones sobre las bases de datos, en realidad se comporta como un interfaz entre las bases de datos y nuestras aplicaciones. La aplicación se comunicarán con el servidor para leer o actualizar las bases de datos.

Por otra parte, trataremos con un lenguaje de consulta y mantenimiento de bases de datos: SQL (Structured Query Language). SQL es un lenguaje en sí mismo, pero mediante el API adecuado podemos usarlo dentro de nuestros propios programas

escritos en otros lenguajes, como C o C++. También desde java incluyendo las librerías adecuadas he podido lanzar consultas directamente en lenguaje sql.

A pesar de la potencia de mysql tiene un inconveniente: su interfaz de trabajo, mediante consola, es bastante engorroso, por lo que trabajé con phpMyAdmin, que presenta una interfaz de trabajo mucho más amigable.
Esta interfaz Web permite administrar el mysql fácilmente con un inconveniente: las claves foráneas no funcionan, lo cual exige poner una especial atención a la hora de manipular las tablas para mantener la integridad de datos. El aspecto de dicha interfaz es el de la figura 5.

Fig. 5 *Interfaz de phpMyAdmin*

Para poder utilizar este programa, hecho en php, era necesario instalar php, apache y mysql por lo que utilicé un pequeño paquete EasyPHP 1.8 que incluía apache 1.3.33, php 4.3.10, mysql 4.1.9 y phpMyAdmin 2.6.1 de fácil instalación y configuración. Este paquete genera un ejecutable desde el que podemos manipular fácilmente el servidor y mysql (ver figura 6)

Fig.6 *Interfaz de EasyPHP con el menú de mysql desplegado*

5.1.2 Servidor (Tomcat)

En el caso de haber trabajo con php el servidor apache nos hubiera servido ya que éste es capaz de interpretar y traducir este código, pero al trabajar con struts necesitaba un servidor contenedor de servlets[6].
Tomcat (Jakarta Tomcat) es un servidor de aplicaciones que sirve como contenedor de Servlets y Java Server Pages[7] (JSP) desarrollado bajo el proyecto Jakarta en Apache Software Foundation.

Tomcat 5.5.x necesita la versión 1.5 del (J2SE) del JDK, Java Development Kit"(JDK),"Standard Development Kit" (SDK) y "Java 2 Standard Edition" (J2SE) son nombres para el mismo componente e incluyen: El API de Java, el JRE (JVM), compilador de Java y otras funcionalidades definidas por Sun.

A pesar de que inicialmente partí de trabajar con estas versiones, por problemas de incompatibilidades y/o colisiones de instalación, tuve que reinstalar todo a mitad de desarrollo y bajar versión a Tomcat 5.0.x (con sus correspondientes versiones compatibles de java y struts).

El servidor Tomcat puede funcionar como servidor Web por sí mismo. En sus inicios existió la percepción de que el uso de Tomcat de forma autónoma era sólo recomendable para entornos de desarrollo y entornos con requisitos mínimos de velocidad y gestión de transacciones. Hoy en día ya no existe esa percepción y Tomcat es usado como servidor Web autónomo en entornos con alto nivel de tráfico y alta disponibilidad.

La estructura de directorios de tomcat es importante ya que al instalarlo se genera una variable de entorno CATALINA_HOME que corresponde al path donde se encuentra el contenedor de servlets. Dicha estructura es:

- bin - arranque, cierre, y otros scripts y ejecutables
- common - clases comunes que pueden utilizar Catalina Catalina y las aplicaciones Web

[6] Ver apéndice de glosario de términos
[7] Ver apéndice de glosario de términos

- conf - ficheros XML y los correspondientes DTD para la configuración de Tomcat
- logs - logs de Catalina[8] y de las aplicaciones
- server - clases utilizadas solamente por Catalina
- shared - clases compartidas por todas las aplicaciones Web
- webapps - directorio que contiene las aplicaciones Web que cargará el navegador cuando haga la petición al servidor
- work - almacenamiento temporal de ficheros y directorios.

En la figura 7 vemos un sencillo esquema de la posición que ocupa el servidor Tomcat dentro de esta arquitectura.

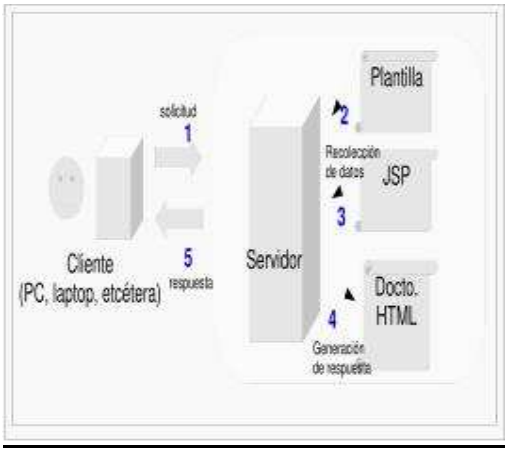

Fig.7 *Esquema de arquitectura con servidor Tomcat*

5.1.3 Lenguaje de programación (java J2EE)

Para poder implementar toda esta arquitectura era necesario trabajar con java. J2EE es una tecnología relativamente novedosa basada en Java. Por esta característica de ser novedoso fue el principal motivo que me impulsó a usarla, ya que consideraba que el concepto de Proyecto Final de Carrera tiene intrínseco una connotación de investigación.

Al ser nuevo no hay demasiada documentación al respecto lo cual ha complicado un poquito (o bastante) más la documentación sobre el tema y el poder trabajar con él.

Java es sobradamente conocido, por lo que no me extenderé en dar detalles, sólo mencionaré algunas de las características principales de este lenguaje que hacen de él que sea tan potente y extendido. Éstas propiedades fueron causa efecto (no sé bien que fue antes) para decantarme por usar la tecnología usada.

- Es multiplataforma, por que el código que he escrito funcionará sobre cualquir sistema operativo que tenga una máquina virtual JAVA.

[8] Ver apéndice de glosario de términos

- Lenguaje de programación orientado a objetos que proporciona mayor capacidad de modularidad

Los programadores de aplicaciones J2EE escriben componentes de aplicación J2EE. Un componente J2EE es una unidad de software funcional auto-contenida que se ensambla dentro de una aplicación J2EE y que se comunica con otros componentes de aplicación. La especificación J2EE define los siguientes componentes de aplicación:

- Componentes de Aplicación Cliente
- Componentes JavaBeans Enterprise
- Componentes Servlets y JavaServer Pages (también llamados componentes Web)
- Applets

5.1.4 Entorno de programación (eclipse)

Eclipse es una plataforma de software de código abierto independiente para desarrollar lo que el proyecto llama "Aplicaciones de Cliente Enriquecido", opuesto a las aplicaciones "Cliente-liviano" basadas en navegadores. Esta plataforma, típicamente ha sido usada para desarrollar entornos integrados de desarrollo (del Inglés IDE), como el IDE de Java llamado *Java Development Toolkit* (JDT) y el compilador (ECJ) que se embarca como parte de Eclipse (y que son usados también para desarrollar el mismo Eclipse). Sin embargo, también se puede usar para otros tipos de aplicaciones cliente, como Bit Torrent Azureus.
Existen muchos plug-in para eclipse aunque hay que prestar especial atención a la compatibilidad de versiones. En mi caso utilicé el plug-in de Lomboz que entre otras cosas me ofrecía la facilidad de replegar (comúnmente llamado deployar[9]) el proyecto y arrancar el servidor desde el entorno eclipse y también la posibilidad de poder depurar el código java.

5.2 Profundizando en Struts

Struts es un proyecto Open-Source creado por la fundación Apache. Es un framework de java basado en la tecnología Java Servlet y en menor nivel, en JavaServer Pages, y, por lo tanto, depende de un contenedor Web. Además Struts ofrece varias funcionalidades requeridas por las aplicaciones más complejas de ambientes Web de Java:

- Diseño de "Templates/Tiles" para aplicaciones tipo Portal
- Utilización de Java Beans
- Servlet Controlador con diversas funcionalidades

[9] Ver glosario de términos en apéndice

- Tags para ambientes JSP's con lógica de ejecución

5.2.1 Conceptos previos

Las primeras especificaciones JSP presentaron dos enfoques para crear aplicaciones Web utilizando la tecnología JSP. Estos dos enfoques eran las arquitecturas JSP Modelo 1 y Modelo 2. Aunque estos términos ya no se utilizan en la especificación JSP, se siguen utilizando en toda la comunidad de desarrollo del nivel Web.

La principal diferencia de las dos arquitecturas está en cómo y qué componentes gestionan en el procesamiento de una petición.

- **Modelo 1**: la página JSP gestiona todo el procesamiento de una petición y es la responsable de mostrar el resultado al cliente. No hay servlets implicados en el proceso. La petición del cliente se envía directamente a una página JSP, que se puede comunicar con JavaBeans[10] u otros servicios., pero la página JSP selecciona la siguiente página para el cliente. La siguiente vista se determina basándose en la JSP seleccionada o los parámetros dentro de la petición del cliente.
- **Modelo 2**: la petición la interpreta primero un sevlet, comúnmente conocido como un servlet controlador. Este servlet gestiona el procesamiento incial de la petición y determina qué página JSP mostrar a continuación (ver *Fig.8*). Como se muestra en la figura un cliente nunca envía un petición directamente a una jsp en la arquitectura de Modelo 2. Esto permite que el servlet lleve a cabo un procesamiento de front-end incluidos, autenticación y autorización, conexión centralizada y ayuda con internacionalización. Una vez que se ha completado el procesamiento de la petición, el servlet dirige la petición a la página JSP apropiada.

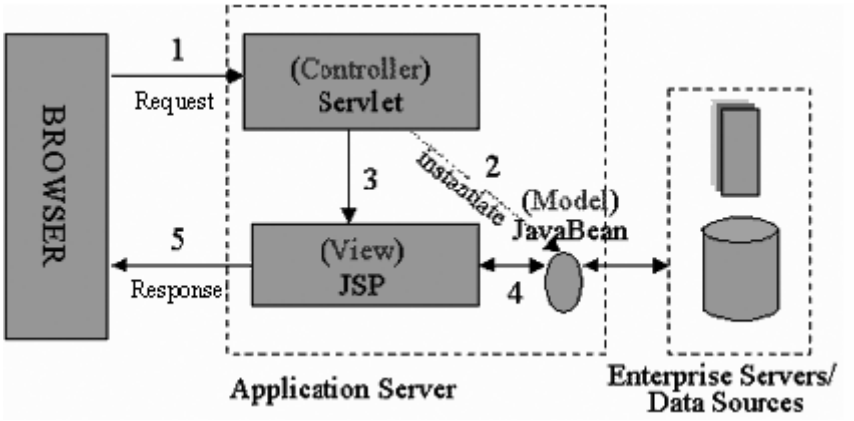

Fig. 8 *Arquitectura JSP Modelo 2*

En resumen, la principal diferencia entre los dos enfoques es que la arquitectura de Modelo 2 introduce un servlet controlador que proporciona un único punto de entrada y permite **más reutilización y extensibilidad** que el enfoque del Modelo 1.

[10] Ver glosario de términos en apéndice

Con la arquitectura de Modelo2, existe una clara separación entre la lógica de negocio, el resultado de presentación y el procesamiento de petición. Esta separación se conoce como un patrón **MVC Modelo-Vista-Controlador**.

5.2.2 Patrón Modelo-Vista-Controlador

El patrón de arquitectura MVC no está directamente relacionado con las aplicaciones Web o Java. De hecho es bastante común en aplicaciones Smalltalk que por lo general no tienen nada que ver con la Web. La justificación de por qué este patrón es tan importante es porque el desarrollo y mantenimiento de la aplicación es mucho más sencillo si los diferentes componentes de una aplicación Web tienen responsabilidades claras y distintas.

Este patrón tiene tres componentes claves (ver *Fig.9*):

- **Modelo**: responsable de la lógica de negocio.
- **Vista**: responsable de una vista de presentación de la lógica de negocio.
- **Controlador**: responsable de controlar el flujo y estado de la entrada de datos por parte del usuario.

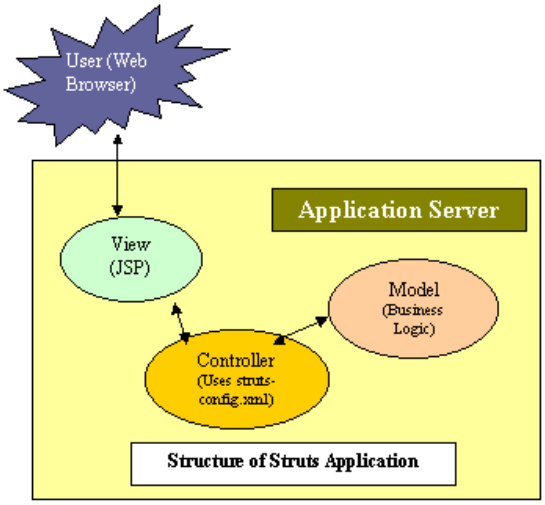

Fig. 9 *Esquema básico de las tres capas de patrón MVC*

Un modelo puede tener diversas vistas, cada una con su correspondiente controlador. Un ejemplo clásico es el de la información de una base de datos, que se puede presentar de diversas formas: diagrama de tarta, de barras, tabular, etc. En el caso de este proyecto los artículos pertenecen a la capa de modelo y tiene diferentes vistas en función de si quien hace la petición es un invitado o una pareja.

Entrando un poco más en detalle:

- El **modelo** es el responsable de:
    - Acceder a la capa de almacenamiento de datos. Lo ideal es que el modelo sea independiente del sistema de almacenamiento.
    - Define las reglas de negocio (la funcionalidad del sistema). Un ejemplo de regla puede ser: "Si la mercancía pedida no está en el almacén, consultar el tiempo de entrega estándar del proveedor".
    - Lleva un registro de las vistas y controladores del sistema
    - Si estamos ante un modelo activo, notificará a las vistas los cambios que en los datos pueda producir un agente externo (por ejemplo, un fichero bacth que actualiza los datos, un temporizador que desencadena una inserción, etc).
    - Si estamos ante un modelo activo, notificará a las vistas los cambios que en los datos pueda producir un agente externo (por ejemplo, un fichero bath que actualiza los datos, un temporizador que desencadena una inserción, etc).

- El **controlador** es el responsable de:
    - Recibe los eventos de entrada(un clic, un cambio en un campo de texto, etc.)
    - Contine reglas de gestión de eventos, del tipo "SI Evento Z, entonces Acción W". Estas acciones pueden suponer peticiones al modelo o a las vistas. Una de estas peticiones a las vistas puede ser una llamada al método "Actualizar()". Una petición al modelo puede ser "Obtener_tiempo_de_entrega(nueva_orden_de_venta)".

- Las vistas son responsables de:
    - Recibir datos del modelo y mostrarlos al usuario.
    - Tienen un registro de su controlador asociado (normalmente porque además lo instancia).
    - Pueden dar servicio de “Atualización()”, para que sea invocado por el controlador o por el modelo (cuando es un modelo activo que informa de los cambios en los datos producidos por otros agentes).

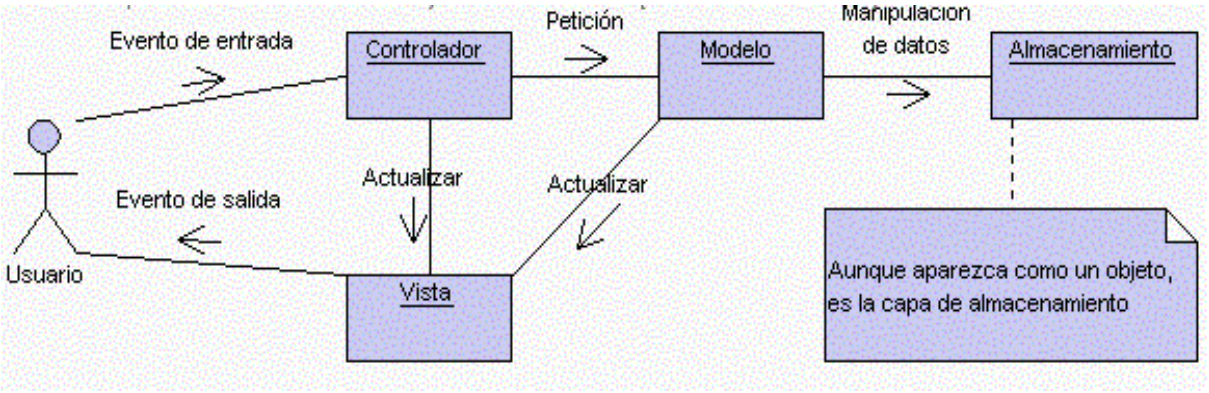

Fig.10 *Gráfico detallado de los componentes modelo vista controlador*

Uno de los argumentos claves contra la utilización del enfoque MVC es que parece extremadamente complicado. Aunque esto podría se cierto a primera vista, utilizar una estrategia MVC puede simplificar el diseño y construcción de la aplicación. Las aplicaciones Web creadas utilizando el enfoque de Modelo2 son, por lo general, más sencillas de mantener que las aplicaciones creadas con la arquitectura de Modelo 1.

5.2.3 Descripción del proceso

El navegador genera una solicitud que es atendida por el Controller (un Servlet especializado). El mismo se encarga de analizar la solicitud, seguir la configuración que se le ha programado en su XML (fichero web.xml) y llamar al Action correspondiente pasándole los parámetros enviados. El Action instanciará y/o utilizará los objetos de negocio para concretar la tarea. Según el resultado que retorne el Action, el Controller derivará la generación de interfaz a una o más JSPs, las cuales podrán consultar los objetos del Model a fines de realizar su tarea.

5.2.4 Configuración

No es el propósito de este apartado detallar todos lo pasos de configuración de struts ya que me extendería demasiado pero si haremos referencia a dos ficheros de vital importancia: web.xml y struts-config.xml

- **web.xml**: cuando el servidor recibe una petición es el primer fichero que va a consultar. En él se detallan comportamientos básicos como cuál será el primer fichero que debe cargar para mostrar, el servlet que atenderá las peticiones, donde está el fichero de configuración (el struts-config.xml), localización de las librerías de tags, etc. Veamos parte de su contenido:

<web-app>

```
<!-- Standard Action Servlet Configuration (with debugging) -->
  <servlet>
    <servlet-name>action</servlet-name>
    <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
    <init-param>
      <param-name>config</param-name>
      <param-value>/WEB-INF/struts-config.xml</param-value>
    </init-param>
    <init-param>
      <param-name>debug</param-name>
      <param-value>2</param-value>
    </init-param>
    <init-param>
      <param-name>detail</param-name>
      <param-value>2</param-value>
    </init-param>
    <load-on-startup>2</load-on-startup>
  </servlet>


 <!-- Standard Action Servlet Mapping -->
  <servlet-mapping>
    <servlet-name>action</servlet-name>
    <url-pattern>*.do</url-pattern>
  </servlet-mapping>


<!-- The Usual Welcome File List -->
  <welcome-file-list>
    <welcome-file>/index.jsp</welcome-file>
  </welcome-file-list>


<!-- Struts Tag Library Descriptors -->
  <taglib>
    <taglib-uri>/tags/struts-bean</taglib-uri>
    <taglib-location>/WEB-INF/tld/struts-bean.tld</taglib-location>
  </taglib>

  <taglib>
    <taglib-uri>/tags/struts-html</taglib-uri>
    <taglib-location>/WEB-INF/tld/struts-html.tld</taglib-location>
  </taglib>

  <taglib>
    <taglib-uri>/tags/struts-logic</taglib-uri>
    <taglib-location>/WEB-INF/tld/struts-logic.tld</taglib-location>
  </taglib>
```

```
  <taglib>
    <taglib-uri>/tags/struts-nested</taglib-uri>
    <taglib-location>/WEB-INF/tld/struts-nested.tld</taglib-location>
  </taglib>

  <taglib>
    <taglib-uri>/tags/struts-tiles</taglib-uri>
    <taglib-location>/WEB-INF/tld/struts-tiles.tld</taglib-location>
  </taglib>

</web-app>
```

La sección:

```
  <servlet-mapping>
    <servlet-name>action</servlet-name>
    <url-pattern>*.do</url-pattern>
  </servlet-mapping>
```

es de vital importancia (junto con la que define la clase que corresponde al servlet llamado acition), en esta sección se le indica al servidor que todas las peticiones que correspondan al patrón **.do* serán atendidas por el servlet llamado action. De este modo siempre que hagamos un

- <a *href="listaBodasAction.do"*>
- *<html:form action="listaBodasAction.do"/>*[contenido de la jsp] submit *</html:form>*
- o desde el fichero struts-config.xml en un mapping[11]

será la clase *org.apache.struts.action.ActionServlet* la encargada de atender la petición.

- **struts-config.xml**: leyendo este fichero con atención, prácticamente podríamos sacar el análisis funcional de la aplicación. Por eso es tan importante la comprensión de este fichero en struts para nuestras aplicaciones. Entre otras cosas se configura el orígen (qué, cómo y dónde) de la base de datos, qué formbeans tenemos, a qué actions están asociados, si son formularios con validación, cuándo y cuál será la proxíma jsp (o action) que deberemos cargar, etc.
  Veamos el contenido de este fichero, que aunque un poco extenso es el eje principal del funcionamiento de struts:

```
<!-- ========================================= Action Mapping Definitions -->


<action-mappings>

<action path="/init" forward="/jsp/login/login.jsp" />
```

[11] Este concepto lo veremos en la explicación del struts-config.xml

```
<action path="/myinit" forward="/jsp/mylogin/mylogin.jsp" />
<action path="/informacion" forward="/jsp/invitados/informacion.jsp" />
<action path="/contactar" forward="/jsp/invitados/contactar.jsp" />
<action path="/opciones_invitados"
        forward="/jsp/invitados/opciones_invitados.jsp"/>
<action path="/iglesia" forward="/jsp/invitados/iglesia.jsp" />
<action path="/restaurante" forward="/jsp/invitados/restaurante.jsp" />
<action path="/lista_bodas_invitados"
         forward="/jsp/invitados/lista_bodas_invitados.jsp" />
<action path="/cuenta_bancaria" forward="/jsp/invitados/cuenta_bancaria.jsp" />
<action path="/dedica" forward="/jsp/invitados/dedica.jsp" />
<action path="/canci" forward="/jsp/invitados/cancion.jsp" />
<action path="/opciones_pareja" forward="/jsp/novios/opciones_pareja.jsp" />
<action path="/ficha_usuario" forward="/jsp/novios/ficha_usuario.jsp" />



        <action
                path="/mylogin"
                type="org.nuvis.control.myLoginAction"
                name="myloginForm" scope="session" >
                <forward name="success_invitado"
                path="/jsp/invitados/opciones_invitados.jsp" />
                <forward name="success_pareja"
                path="/jsp/novios/opciones_pareja.jsp" />
                <forward name="error" path="/jsp/mylogin/myerror.jsp" />
         </action>

           <action path="/login"
                type="org.nuvis.control.LoginAction"
                name="loginForm" scope="session">
                        <forward name="success" path="/jsp/login/display.jsp" />
                        <forward name="error" path="/jsp/login/error.jsp" />
           </action>


           <!-- Action q redirige a pagina de confirmacion -->
           <action path="/confirma"
                type="org.nuvis.control.ConfirmAction"
                name="confirmForm" scope="session"
                input="/jsp/invitados/confirma.jsp">
                        <forward name="success" path="/jsp/invitados/confirma.jsp" />
                                <forward name="error" path="/jsp/login/error.jsp" />
           </action>

                <!-- Action ejecutado al aceptar confirmación -->
                <!-- OJO el valor del input es donde va si hay error en el formulario -->
                <action path="/confirmar"
                        type="org.nuvis.control.ConfirmarAction"
```

```
            name="confirmForm" scope="session">
                <forward name="success"
                path="/jsp/invitados/confirmada.jsp" />
                <forward name="error" path="/jsp/login/error.jsp" />
</action>

<action path="/dedicatoria"
    type="org.nuvis.control.DedicaAction"
    name="dedicaForm" scope="request" >
        <forward name="success" path="/jsp/invitados/dedicada.jsp" />
        <forward name="error" path="/jsp/login/error.jsp" />
    </action>

    <action path="/grabaCancion"
        type="org.nuvis.control.GrabaCancionAction"
        name="cancionForm" >
        <forward name="success"
        path="/jsp/invitados/cancionada.jsp"/>
        <forward name="error" path="/jsp/login/error.jsp" />
    </action>

    <action path="/lista_canciones_invitados"
    type="org.nuvis.control.ListaCancionesAction" scope="request" >
       <forward name="success"
            path="/jsp/invitados/lista_canciones.jsp" />
       <forward name="error" path="/error.jsp" />
    </action>

    <!--   PARTE NOVIOS   -->
    <action path="/lista_canciones"
    type="org.nuvis.control.ListaCancionesAction" scope="request" >
       <forward  name="success"  path="/jsp/novios/lista_canciones.jsp"  />
       <forward name="error" path="/error.jsp" />
    </action>

    <action path="/lista_dedicatorias"
    type="org.nuvis.control.ListaDedicatoriasAction" scope="request" >
       <forward name="success"
            path="/jsp/novios/lista_dedicatorias.jsp" />
       <forward name="error" path="/error.jsp" />
    </action>

    <action path="/lista_bodas_novios"
        type="org.nuvis.control.ListaBodasAction" scope="request" >
            <forward name="success"
            path="/jsp/novios/lista_bodas_novios.jsp" />
            <forward name="error" path="/jsp/login/error.jsp" />
    </action>
```

```
<action path="/ficha_articulo"
    type="org.nuvis.control.FichaArticuloAction"
            name="articuloForm" scope="request" >
            <forward name="success"
            path="/jsp/novios/ficha_articulo.jsp" />
            <forward name="error" path="/jsp/login/error.jsp" />
</action>

<action path="/lista_invitados"
    type="org.nuvis.control.ListaInvitadosAction" scope="request" >
       <forward name="success" path="/jsp/novios/lista_invitados.jsp" />
       <formard name="error" path="/error.jsp" />
</action>

<action path="/elimina_invitado"
    type="org.nuvis.control.EliminaInvitadoAction" scope="request" >
            <forward name="success" path="/lista_invitados.do"/>
            <forward name="error" path="/error.jsp" />
</action>

<action path="/graba_usuario"
    type="org.nuvis.control.GrabaUsuarioAction"
    name="usuarioForm" scope="request" >
            <forward name="success"
            path="/jsp/novios/ficha_invitados.jsp"/>
            <forward name="error" path="/error.jsp" />
            <forward name="ok" path="/jsp/novios/opciones_pareja.jsp" />
</action>

<action path="/graba_invitados"
    type="org.nuvis.control.GrabaInvitadosAction"
    name="usuarioForm" scope="request" >
            <forward name="success" path="/lista_invitados.do"/>
            <forward name="error" path="/error.jsp" />
</action>

    <action path="/graba_articulo"
            type="org.nuvis.control.GrabaArticuloAction"
                    name="articuloForm" scope="request" >
                    <forward name="success"
                    path="/lista_bodas_novios.do" />
                    <forward name="error" path="/error.jsp" />
    </action>

    <action path="/elimina_articulo"
            type="org.nuvis.control.EliminaArticuloAction" >
                    <forward name="success"
                    path="/lista_bodas_novios.do" />
                    <forward name="error" path="/error.jsp" />
```

```
        </action>

      <action path="/bbddconnect" type="org.nuvis.db.TestDataSourceActiom" >
        <forward name="success" path="/jsp/test/succes.jsp" />
      </action>

</action-mappings>
```
[12]

5.3 Codificación[13]

Ya tenemos clara cuál será toda la plataforma sobre la que trabajaremos es el momento de empezar a utilizarla. Como no es la pretensión plasmar aquí todo el código de los ciento veinte ficheros aproximadamente necesarios para crear esta aplicación, sólo haré algunos comentarios en los apartados que considere de interés.

5.3.1 Base de datos

Teniendo en cuenta el modelo E/R del *apartado 3.3.1* la construcción de la base de datos se ajustó bastante a la regla de una tabla por cada objeto. Los atributos de cada tabla se corresponden con los atributos del diagrama de clases del *apartado 3.4.*
A la hora de diseñar el modelo de datos hubo que tener en cuenta que era importante construir un modelo que permitiera soportar las situaciones que se dan en la vida real.
En el caso de los *usuarios invitados y parejas*, en un principio se pensó en crear una clase única *usuario* pero había varios problemas:

- un mismo usuario podía a la vez ser pareja de su propia boda e invitado de otra
- un usuario podía pertenecer a dos bodas por lo que había que poder distinguir a que boda pertenecía
- existen datos que sólo serían necesarios en caso de ser pareja o en caso de ser invitados por lo era innecesario tener una tabla en base de datos con muchos campos que en realidad no eran servibles

Otro aspecto a comentar es que la contraseña inicialmente se creó en la tabla de usuarios como tipo *int,* pero se cambió a *string* para contemplar la posibilidad de contraseñas alfanuméricas.

[12] Lamento la indentación mostrada que no es la original, pero se ha tenido que reajustar (o desajustar, más bien) debido a que había instrucciones o etiquetas que no cabían en una línea.

[13] En el apartado de tecnología/struts la explicación de los ficheros struts-config.xml y web.xml está hecha usando los ficheros reales de este proyecto por lo que no volveré a explicarlo en el apartado de codificación

5.3.3 Páginas JSP

En nuestro entorno las JSP's son lo que en una aplicación Web estática serían las páginas html.

5.3.3.1 De html a jsp

A pesar de que las páginas mostradas son jsp en su totalidad para hacerlas el primer paso fue diseñar en papel la rejilla de <tr> y <td> y darles la forma e incluirles los componentes (selects, buttons, etc.) una vez conseguido el resultado deseado el siguiente paso fue transformar los tags de html a tags de struts y darle funcionalidad a la página llamando a los action necesarios, rellenando información proveniente de los beans, o bien de atributos de request, etc. Lo cual me ha resultado bastante complicado dado mi total desconocimiento en toda la materia.

5.3.3.2 Hojas de estilo (CSS Cascade Style Sheets)

Y ya que estaba puesta en complicarme la vida, el aspecto de las jsp no fue logrado a través de dar formato en cada etiqueta y/o componente de la jsp, sino que me hablaron de las hojas de estilo y las apliqué.
CSS es una potente herramienta en especial cuando la aplicación sigue un formato parecido a través de la navegación ya que se basa en crear estilos a modo de clase pudiendo crear estilos "heredados", es como crear subclases. Por ejemplo al tag *<td>* lo dotamos de un estilo determinado y al tag *<td>   </td>*, así el primero cogería el estilo definido para *td* y el segundo para *td.a* que seria diferente del creado para *a.*

5.3.4 Tomcat

Para cargar localmente la aplicación, con una correcta configuración de los ficheros antes descritos, bastaría con poner en nuestro navegador http://localhost:8080/carpetaProyecto pero en mi caso, como está Web llegó a salir a "producción" tuve que añadir un pequeño fichero javaScript para que redireccionara las peticiones recibidas por el puerto 8080 a mi *carpetaProyecto.*
De lo contrario cada vez que alguien hubiera puesto en su navegador la dirección www.nuvis.es lo que el servidor (mi ordenador de casa) hubiera mostrado sería la página index.jsp de tomcat.

5.3.5 Generación de ficheros logs

Al principio de la fase de desarrollo, por desconocimiento y problemas con la instalación de nuevos plug-in no podía depurar por lo que mi método para poder encontrar errores era recurrir a los populares *chivatos*. No hace falta decir que no es un sistema muy

ortodoxo ni profesional.
En el mundo laboral las aplicaciones normalmente van provistas de un sistema de traceado. La ejecución de los programas genera unos ficheros, normalmente con la extensión *.log,* que no son más que chivatos con la información necesaria en caso de incidencias.
Me comentaron la existencia de una librería llamada *log4j-1.2.8.jar* y la apliqué.
Para poder utilizar esta librería hubo que, entre otras cosas, modificar el fichero web.xml que antes mencionaba, tan importante para la aplicación, y añadir:

```
<!--  Configuracion servico logs (arracar al iniciar sevidor) -->
  <servlet>
        <servlet-name>log4j-init</servlet-name>
        <servlet-class>org.nuvis.utils.Log4jInit</servlet-class>
        <init-param>
              <param-name>log4j-init-file</param-name>
              <param-
value>C:/java/eclipse/workspace/nuvis/webapps/config/log4j.properties</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
  </servlet>
```

también tuve que sincronizar la inicialización en todas aquellas clases en las que quisiera utilizar el servicio de logs y dentro de ellas para utilizarlo:

log.info(string), log.debug(string) o log.error(string)

El resultado es un fichero de logs con la información del tipo:

```
2007-06-24 19:35:11,531 [nuvis.control.myLoginAction:91] INFO  nuvis - Ha inicialidado
datasource, connection, stmt y rst
2007-06-24 19:35:11,546 [nuvis.control.myLoginAction:100] DEBUG nuvis - Password
validado. Usuario y password metidos como atributos sesion
2007-06-24 19:35:11,546 [nuvis.control.myLoginAction:113] INFO  nuvis - le corresponde
usuario1
2007-06-24 19:35:11,546 [nuvis.control.myLoginAction:160] INFO  nuvis - Conexion
cerrada en finally
2007-06-24 19:35:11,546 [nuvis.control.myLoginAction:171] INFO  nuvis - myLoginAction:
OK
2007-06-24 19:35:11,546 [nuvis.control.myLoginAction:174] DEBUG nuvis - usuario pareja
2007-06-24 19:36:53,203 [nuvis.control.myLoginAction:60] INFO  nuvis - Dentro del
myLoginAction
```

Capítulo

6 Objetivos vs. Resultados

Desde la plataforma sobre la que construir inicialmente pensada, hasta la funcionalidad pasando por el diseño de pantallas, han cambiado de la idea con la que se partió al resultado finalmente obtenido.

Inicialmente se pensó en crear un proyecto sobre LAMP (Linux, apache, mysql y php), sin embargo esta idea de partida fue cambiando en la fase de especificación a medida que el proyecto iba madurando.

El sistema operativo linux, aunque potente me limitaba la máquina sobre la que poder trabajar: sólo el ordenador de casa, ya que no es un sistema operativo muy extendido en el mercado y esto me impedía poder utilizar el portátil del trabajo para desarrollar o modificar cosas fuera de casa. Por este motivo decidí pasarme a Windows y este cambio llevó a plantearme otros posibles cambios que se sucedieron hasta llegar a la plataforma finalmente usada que ha sido Windows, Tomcat (también de apache), mysql y J2EE trabajando en el entorno de desarrollo de eclipse usando el framework de J2EE de struts.

Inicialmente se pensó en unas funcionalidades que no se han llevado a cabo:

- un tercer perfil de usuario: el usuario anónimo. Inicialmente se creía conveniente que existiera este perfil que se correspondía con un usuario que no estaba invitado a la boda pero que por cercanía le pudiera interesar entrar en la Web y tener acceso a algunas funcionalidades. Las opciones para este tipo de usuario iban a ser las más restrictivas: poder dejar dedicatorias y lista de regalos y poco más, pero el cliente consideró finalmente que era poco ético.

- el sistema de recomendación de canciones en un inicio se concibió para que el invitado subiera físicamente la canción al servidor, pero por cuestiones legales y de espacio en el servidor finalmente se pensó en simplemente dar la información para que fueran los novios o la empresa que organiza la música de la boda, quienes se encargaran de proporcionar la canción al discjockey.

- la funcionalidad de la lista de bodas se había pensado para que funcionara tipo “carrito”. Los invitados irían añadiendo artículos a su carrito y finalmente podrían pagar, ingresando en la cuenta de los novios, el importe total de lo seleccionado. Pero esta idea inicial también cambió: por un lado parecía fría que se redujera a hacer un ingreso y por otro lado surgía el inconveniente de tener que entrar en temas de licencias y de contratación de TPV.

  Una posible solución hubiera sido incorporar en nuestro proyecto BYPAL incorporando sólo el trozo de código para su utilización y esta empresa y/o funcionalidad ya incorpora sus licencias y métodos de encriptación para seguridad y confidencialidad de datos. Ésta sería también una posible mejora aunque en nuestro caso optamos por aprovechar la infinidad de páginas comerciales desde las que se podía comprar, así uno de los campos del artículo en la dirección url que

nos enlaza directamente con la página del artículo desde la que sí que podemos comprar.

Con esta alternativa solucionábamos el tema de seguridad y además, el invitado realmente compraba un regalo y no se limitaba a hacer un ingreso de dinero, que al cliente le resultaba más frío.

Por otro lado ha habido muchas opciones que finalmente se han incorporado y que en un principio no existían como la opción de dejar dedicatorias, o mostrar información adicional al consultar determinados datos.

Exceptuando estas pequeñas variaciones en cuanto a las opciones disponibles y/o alguna variación respecto a su funcionamiento, se puede decir que los objetivos iniciales se corresponden bastante con el resultado finalmente obtenido.

Bueno, si el tiempo es un factor a contar como objetivo inicial, éste también sería un punto a incluir en esta sección ya que inicialmente preví que me llevaría un cuatrimestre y finalmente han resultado ser dos, y porque llega un momento en el que tienes que trazar la línea de fin porque a medida que vas trabajando e implementando se te van ocurriendo nuevas funcionalidades y ampliaciones posibles que no tendrían fin.

Capítulo

7 Conclusiones

Para llegar a un producto que cumpliera las expectativas del cliente, fueron necesarias varias reuniones con el cliente, con la intención de determinar definitivamente cuáles eran las funcionalidades que requería del sistema,

Estas entrevistas iban acompañadas de prototipos básicos, pero aclaradores. Quizás una vez visto el resultado cualquier compañero juzgaría que éstos eran bastante claros y fáciles de definir, pero el problema del ingeniero es que tiene que traducir unas necesidades de la calle, sobre un tema no siempre conocido o familiar, con lenguaje no informático, a acciones o funciones que viables.

Esta fase, al inicio parecía ser la más fácil, aparentemente trivial, sin embargo llevó bastante tiempo y realmente no se llegaron a pulir expectativas hasta que el cliente vio la aplicación en funcionamiento. Ahora, ya al final, y con la capacidad de echar una mirada retrospectiva me atrevo a afirmar que ha sido una de las fases, si no complicadas, más largas de superar.

En cuanto al desarrollo de la aplicación en sí, de no ser porque estoy en la recta final, con las horas acumuladas que conlleva eso, volvería hacerlo de nuevo. Debe sonar muy rara esa afirmación pero tiene una explicación: cuando inicié el proyecto no tenía ni idea de lo que era una hoja de estilo, una jsp, el famoso struts, ni Tomcat ni nada, con lo cual ha sido un “bocado” bastante grande el que me he llevado a la boca que no ha sido fácil de digerir.

Este absoluto desconocimiento inicial de la materia ha supuesto que toda la fase de desarrollo fuera un continuo proceso de aprendizaje y autoformación, lo que empezó siendo un prueba y error acabó siendo algo que entendía. Por lo que me daba cuenta que si volviera a empezar habría muchas cosas que haría de forma diferente, más estructuradamente.

Struts ofrece una potencia de uso que probablemente no habré utilizado ni al 10%, pero para haber sido mi primer contacto y haberlo odiado en muchos momentos, a día de hoy puedo afirmar que me alegro de haberme embarcado en esta locura y haber sufrido lo que he sufrido a cambio de tener los mínimos conocimientos que tengo hoy.

Capítulo

8 Trabajos Futuros

Dada la característica de ser un sistema adaptado a las necesidades de los novios, así como a las posibles sugerencias o necesidades surgidas por invitados, los trabajos futuros son casi infinitos e ilimitados, tantos como nuevos requerimientos vayan surgiendo, bien por necesidad bien debidos a la evolución y a las nuevas modas o tendencias que puedan surgir entorno a este mundo de las bodas u otras celebraciones.

8.1 Factibilidad de ampliación

Ya en la fase de viabilidad del proyecto, ya se vio que el campo de aplicación de este sistema podía ser mucho más amplio del que lo es en la actualidad. Fue pensando en esas posible ampliaciones por lo que la plataforma utilizada se cambió de *php* a *struts* en la que el código es reutilizable con flexibilidad y el proyecto resultante ofrecía una fácil escalabilidad.

8.2 Ampliaciones generales

Dado que la estructura de la base de datos se puede aplicar a otro tipo de celebración, la aplicación podría subir un nivel de abstracción en la que indicáramos el tipo de ceremonia y aplicarlo tanto a comuniones, bautizos, bodas de plata, cenas de compañeros de primaria, etc. En general sería aplicable a todas aquellos eventos en los que se necesita reunir a una cantidad importante de personas, ahorrándonos así las mil y una llamadas solicitando confirmación y dando datos sobre dónde y cuándo.

Para poder llevar a cabo este salto, en realidad, gracias a la estructura que hay montada en la actualidad, tan sólo sería necesario agregar un atributo al objeto *pareja* (que en la situación de mayor abstracción se entendería como el *celebrante*) en el que indicáramos el tipo de celebración.

Éste atributo sería recogido por el sistema para saber qué mensaje de bienvenida mostrar y qué opciones debía ocultar y cuáles dejar como confirmar *asistencia, información, contactar*, etc.

Como vemos, realmente el campo de aplicación es mucho mayor del que a simple vista parece.

8.3 Ampliaciones concretas al proyecto actual

El cliente no especificó que la aplicación estuviera abierta a la posibilidad de que varias parejas trabajaran con ellas, sin colisión ni pérdida de confidencialidad alguna, ya que pensó en su caso concreto, su necesidad actual, aunque cuando se le planteó vio bien la propuesta. De este modo se conseguía un sistema más potente y abierto.

En la actualidad es el administrador el que da de alta una nueva pareja aunque, para conseguir completar el objetivo inicial de que fuera un sistema autogestionable, sin necesidad de un administrador, sería necesario agregar una opción en la página de inicio que ofreciera la posibilidad de darse de alta en el sistema como nueva pareja: *registrarme como pareja* esa opción llamaría a un *action* que mostraría una *jsp* como muestra la figura *(*Ver *Fig. 2)*.

Los datos de este formulario se corresponderían prácticamente en la totalidad con los atributos del objeto *pareja.*

Aunque en la actualidad no se está utilizando, la base de datos está prepara para almacenar los e-mails de forma fraccionada, es decir no existe un campo que almacene la dirección de correo completa sino que el mail ser fraccionaría en dos substrings para ofrecer mayor seguridad al sistema contra intrusiones.

Fig. 2 Ejemplo aproximado de formulario requerido para el alta de pareja

Otra de las ampliaciones que considero sería interesante incorporar al sistema, sería codificar un motor generador de contraseñas para que automáticamente al crear usuarios o parejas el sistema asignase una por defecto, a lo que, consecuentemente, también habría que añadir la funcionalidad de *cambio de contraseña*. Y para conseguir un aspecto más profesional completaríamos esta área ofreciendo otra opción añadida de *recordar*

contraseña.

Hay una funcionalidad que no se le ocurrió al cliente pero que, con la experiencia de haber pasado por el periodo de uso de la aplicación, hemos considerado que hubiera sido de una gran utilidad: añadir una opción al perfil de novios de *estadísticas.* Esta operativa nos llevaría a una página en la que tuviéramos un resumen de datos: total personas confirmadas, total sin confirmar, totales plazas autocares, total regalos disponibles, etc.

De esta forma tendríamos toda la información de interés a nivel de organización de forma clara.

Ya en otro nivel de dificultad, pero de gran utilidad a la hora de organizar una boda, sería ideal que, ya que tenemos la lista de invitados de forma individual y también agrupados por núcleos familiares o parejas, acoplarle una aplicación que recuperara estos nombres y los tratara como “fichas” que pudiéramos arrastrar sobre un esquema de mesas para colocarlos y determinar la ubicación de los invitados en el salón, algo que es realmente un quebradero de cabeza.

También se pensó en un apartado de *álbum de fotos,* en el que se fueran añadiendo fotos según la etapa, que pudieran resultar de interés común para todas aquellas personas que asisten al evento (fotos de los novios con diferentes invitados, fotos de la despedida, fotos de la boda, etc.)

Otra posible vía, por la que ampliar el actual sistema, sería dotar al sistema de un foro

Éstas son solo algunas de la múltiples ampliaciones y vías de continuación; con lo que nos hacemos una idea de que este proyecto es tan sólo la punta del iceberg de lo que puede llegar a ser.

Capítulo

9 Apéndice

9.1 Glosario de términos

requisito: en lenguaje informático descripción formal del comportamiento externo del sistema des un punto de vista del usuario o entorno. Su función es explicar qué hace el sistema-

framework: palabra adoptada del inglés, se traduciría por *marco de trabajo*. Es la extensión de un lenguaje mediante una o más jerarquías de clases que implementan una funcionalidad y que (opcionalmente) pueden ser extendidas. Puede involucrar TagLibraries (*librerías de etiquetas*).

modelo Entidad/Relación: (Chen 1976[14]) es un modelo de datos semántico cuyo objetivo inicial era vencer algunas de las dificultades mostradas por el modelo relacional, al que pretendía sustituir. Concretamente, pretendía dotar de "significado" a las estructuras de datos, carentes del mismo, del modelo relacional

java server pages (JSP): La JSP es una tecnología Java que permite a los programadores generar contenido dinámico para Web, en forma de documentos HTML, XML o de otro tipo. Las JSP's permiten al código Java y a algunas acciones predefinidas pueden ser incrustadas en el contenido estático del documento web.

DTD: definición de tipo de documento (DTD) es una descripción de estructura y sintaxis de un documento XML.

servlet: son objetos que corren dentro del contexto de un contenedor de servlets (ej: Tomcat) y extienden su funcionalidad. También podrían correr dentro de un *servidor de aplicaciones* (ej: OC4J Oracle) que además de contenedor para servlet tendrá contenedor para objetos más avanzados como son los EJB (Tomcat sólo es un contenedor de servlets).

Catalina: es el nombre del contenedor de servlets del Jakarta Topmcar desde la versión 4x. Fue desarrollado bajo el Proyecto Jakarta de la Apache Software Foundation. Tomcat implementa las especificaciones de Sun Microsystems para servlets y JSP’s, las cuales son importantes tecnologías web basadas en Java.

deployar: anglicismo. Palabra utilizada entre programadores para referirse al término replegar. Las aplicaciones web son primero replegadas en un archivo .war y después es el servidor el encargado de desplegarlas cuando recibe una petición.

[14] **Chen, P. P.** (1976) "The Entity-Relationship Model -Toward a Unified View of Data", *Transactions on Database Systems,* Vol. I, No1, March 1976: 9-36.

javaBean: es una forma de modularizar el uso de datos en una aplicación con JSP's/servlets a través de una clase. Su característica principal es el uso de los métodos get y set.

9.2 Bibliografía

CAVANESS, CHUCK. Jakarta Struts / Programming Jakarta Struts. Madrid: Anaya Multimedia, 2005.

CERRADA, J. A.; COLLADO, M.; ESTÍVARIZ, J. F. y GÓMEZ, S. R. Introducción a la Ingeniería de Software. Editorial Centro de Estudios Ramón Areces, S. A. Madrid, 2000.

ANEXOS

A.2 Prototipo navegable

La primera aproximación al proyecto fue la entrevista con el cliente para conocer qué necesitaba cuáles eran sus necesidades y establecer los requisitos.

Una vez los requisitos formalizados se hicieron una serie de prototipos navegables de diseño rápido, sin funcionalidad, para acercarnos más empírica y gráficamente a las verdaderas necesidades del cliente, ya que en muchas ocasiones el mayor problema de un ingeniero es conseguir obtener del cliente toda la información necesaria acerca de la funcionalidad deseada.

Uno de estos prototipos navegables es el documento PowerPoint nuvis.ppt entregado como anexo2.